# Some Thoughts On Multicore

Michael A Heroux
Sandia National Laboratories

## Introduction

Multicore processors appear to be on the radar of almost every software engineer. In a brief time span we have gone from serial-only nodes being dominant to multicore nodes being so. Amazingly the software industry as a whole seems largely unprepared for the required changes. Specifically, in order to begin riding the commodity performance curve again, an application must be able to exploit multicore parallelism in some fashion.

In this paper we briefly present two different but complementary topics that may be of interest to those tasked with exploiting multicore machines.

1. **Exploiting SMP nodes**: Lessons learned from the introduction of shared memory parallel (SMP) nodes into the high performance computing (HPC) community about a decade ago.
2. **Useful software abstractions**: Use of interfaces to define the parallel machine and its operations, along with important abstract entity classes.

## Topic 1: Lessons Learned from SMP Computing

About 10-12 years ago, several computer system vendors introduced commercial SMP nodes that were combined via an interconnect network to create leading edge supercomputers. Although certainly not a complete analogue to our current situation with multicore chips, there are lesson to be learned from these systems. Most notable of these systems, in my opinion, were the SGI Origin and the IBM SP series. HPC application programmers, who had largely adopted a single program multiple data (SPMD) programming model and who were already accustomed to distributed memory parallel (DMP) programming via the Message Passing Interface (MPI), needed to determine how to best exploit these SMP nodes.

Three options were available:

1. **SMP:** Using shared memory only techniques such as Pthreads or OpenMP were an option on the SGI Origin systems since they supported a cache-coherent non-uniform memory architecture even for very large systems. However, the performance impact of most efforts in this direction was initially very poor. Furthermore, the programming model was not portable to other scalable systems, which did not have a global address space.
2. **DMP:** An attractive model was to simply map an MPI process to each processor on a node. This required no change to the application. However, again initial performance results were poor, primarily be the underlying MPI libraries were not “SMP-aware” in the sense that messages sent between processors on a node were not short-circuited, leading to unnecessary bandwidth bottlenecks.

3. **Hybrid SMP under DMP:** Many application developers studied the use of both programming models, partitioning the data at a coarse level for execution across MPI processes, but then using SMP techniques to exploit multiple processors on each node.

Although there are certainly some notable exceptions, the general conclusions from the body of research performed at this time were as follows.

1. SMP only conclusions.
    a. SMP approaches require as much or more effort to produce scalable application performance as DMP only. The basic issues of work and data placement are inherent to SPMD programming.
    b. Issues of false cache-line sharing and poorly conceived incremental strategies to parallelizing codes actually made scalable SMP application development more expensive in many cases.
2. DMP only conclusions.
    a. DMP approaches do work. However, an SMP-aware MPI is required and users should not ask for all processors on a node, e.g., ask for 15 of 16.
    b. Using DMP only is very attractive for existing MPI applications because it required little or no code change, just better MPI libraries.
3. SMP under DMP conclusions.
    a. SMP under DMP can work. However, there must be something in the application that takes advantage of the fact that fine grain data access can be done across processors on an SMP node.
    b. Simply using SMP as a simple replacement for more MPI processes is not effective, and greatly complicates the programming model for applications.

**Summary of SMP vs DMP:** After much effort to determine how SMP nodes could be best utilized in a large scale HPC system, the vast majority of applications developers determined that DMP (MPI) only was the best choice. Once MPI library developers improved their MPI implementations for SMP based systems, there was little incentive to adopt the complicated SMP under DMP programming models. The only exceptions were for applications where a shared memory algorithm could be used within an SMP node and MPI used across nodes.

## Topic 2: Useful Software Abstractions in the Petra Object Model

The Trilinos Project provides a large collection of equation solvers for large scale parallel applications. It also provides an object oriented data model for forming and using matrices, graphs, vectors and other linear algebra objects. Written primarily in an object oriented C++ style, Trilinos makes extensive use of interfaces to access external services and to provide extensibility for users who want to incorporate their own custom software. For the purposes of our discussion, we focus on the Petra Object Model (POM) which, via the Epetra and Tpetra packages in Trilinos, provides data services to the rest of Trilinos. Specifically, we focus on the following four collections of interfaces in POM:

1. Comm: The abstract parallel machine interface. All access to information about, and services from, the parallel machine are done through this interface.
2. ElementSpace: Description of the layout of an object on the parallel machine. All data objects are associated with one or more ElementSpace and are often created using an ElementSpace object.
3. DistObject: A base class for all distributed objects. Used to provide gather/scatter services required to redistribute existing distributed objects.
4. Coarse grain linear algebra objects: Although Trilinos provides many concrete linear algebra objects, each concrete class inherits from one or more abstract linear algebra class and all Trilinos solvers access linear algebra objects via the abstract interfaces. In particular, matrices and linear operators are accessed this way.

Because Trilinos accesses external services and coarse grain linear algebra objects via abstract interfaces, we can introduce special Comm adapters for new parallel machine architectures, and provide specialized linear algebra classes to take advantage of the particular features of the parallel machine. Using the POM in principle we can take advantage of many types of special hardware.

**Summary of Useful Abstractions:** The combination of the above abstract interfaces has been used to support hybrid SMP under DMP programming without changes to any classes except special data classes that are specifically designed to take advantage of the hardware. In other words, the majority of Trilinos software is unaware of the architecture on which it is running. Careful use of interfaces such as POM in other software frameworks can make large applications far more tolerant to hardware and even programming model changes. With POM we have designed selective support for UPC and are considering special linear algebra kernel sets for ClearSpeed processors and Cell.

## Final Summary

Although multicore processors are definitely a new opportunity and challenge for software engineers, there are many things to learn from the HPC community experience with SMP nodes. Furthermore, although many people find MPI programming difficult, well designed MPI applications actually have very little direct dependence on MPI. In fact, if Trilinos is indicative, very few developers on a project team need to have specific knowledge of MPI in order to achieve scalable performance. For this reason, MPI should not be dismissed as a viable option for programming multicore processors, especially if we develop “multicore-aware” implementations of MPI and access capabilities through abstract interfaces as is done in Trilinos. Finally, regardless of which programming model or models are used within an application, effective abstract layers can greatly reduce the impact of adapting to other models and new hardware.